<https://btholt.github.io/complete-intro-to-react-v5/pure-react>

React has two type of components:

1. Function component
2. Class component

Function components must return markup

-Inside a render function, any sort of state cannot be modified because you don’t know when and how function will be called so it can’t be modify any ambient state

(if you update state inside render function then it will stuck inside infinite loop.)

React.createElement creates one instance of some component (creates instance of class- Here App is class and creating an instance)

In createElement last two parameters are optional

ReactDOM.render – takes our rendered App and puts it in DOM

**Npm**

Package Manager for Node- It has all the packages in the front-end scene

npm allows you to bring in code from the npm registry which is a bunch of open-source modules that people have written so you can use them in your project

**Eslint**

**Parcel** is new bundler for JavaScript projects, it will accept entry point and crawl through all its dependencies and output single file

Install Parcel npm install -D parcel-bundler.

Now inside of your package.json put:

"scripts" {

"dev": "parcel src/index.html"

}

We are giving entry point to run. It reads index.html and finds its dependencies

**JSX**

Translates HTML tags to React.createElement calls

{props.name} syntax: this is how you output JavaScript expressions in JSX.

**Notes:** you still have to import **React,** though it’s not explicitly being used. Because JSX is compile to React.createElement calls. Anywhere you use JSX, React need to be imported

Note: P in Pet is capitalized. If it is smaller case React takes it as html component not as React component

**ESLINT+ React**

This particular configuration has a lot of rules to help you quickly catch common bugs but otherwise leaves you to write code how you want.

* The import plugin helps ESLint catch commons bugs around imports, exports, and modules in general
* jsx-a11y catches many bugs around accessibility that can accidentally arise using React, like not having an alt attribute on an img tag.
* react is mostly common React things, like making sure you import React anywhere you use React.
* babel-lint allows ESLint to use the same transpiling library, Babel, that Parcel uses under the hood. Without it, ESLint can't understand JSX.
* eslint-plugin-react now requires you to inform of it what version of React you're using. We're telling it here to look at the package.json to figure it out.

**Hooks**

They let you use state and other React features without writing a Class

Hooks allow you to reuse stateful logic without changing your component hierarchy, which helps to easily share Hooks among components

useState is like creating a hook

Brian explains why two-way data binding isn't free in React, and why this makes the code more maintainable. The useState hook is utilized to update the state, and the onChange event handler is tasked with displaying it in the component.

Shape, rectangle

Description automatically generated

On changing location, kicks off re-render, search parameter re-renders then we have two location and setlocation.

When you get back hook you get array of things:

1. First one is always current state of it(location in below code)
2. Updater function for that particular piece of state(setLocation in code)

Every time you type in location, an event happens (e is event happened in location) then we call setLocation of whatever is inside of that particular input

‘Seattle’ is default state and will be updated if changed.

Graphical user interface

Description automatically generated with medium confidence

Whenever input is changed setLocation is called with whatever is in input and location is going to be Whatever we updated location to be

Render functions run whenever something is updated, to avoid

Note: ALL HOOKS BEGIN WITH ***Use***

We can make custom hooks.

**Best Practices for Hooks**

They never go inside if statements and for loops(Why?) Graphical user interface, text, application

Description automatically generated

Graphical user interface, website

Description automatically generated

Ex(why hooks should not be inside loops): In above code

If something is false- --this is going to get animal instead of location as

Graphical user interface, text, application

Description automatically generated

Rules-of-hooks

0: turn off

1: warn

2: make it an error

Note: useState can be used as many times as we want for different states. Ordering is important as React relies on useState to be called in strictly the same order every time so it can give you the same piece of state.

JSX doesn’t allow just onChange, there has to be onBlur with onChange

useDropdown.js

* Is a reusable hook

We need to install client library to make requests against an API.

import { ANIMALS } from "@frontendmasters/pet";

From above state parcel installs "@frontendmasters/pet” and rebuilds

ANIMALS- Array of strings

{ANIMALS.map(animal **=>** (

<option key={animal} value={animal}>

{animal}

</option>

))}

With ARROW functions – we can have implicit return – above code: arrow function and no curly is implicit return

Below code is same as above

If we have parenthesis, which means continue on next line means implicit return.

{ANIMALS.map(animal **=>** {

<option key={animal} value={animal}>

{animal}

</option>

})}

Custom Hook:

useDropDown.js is custom hook

return [state, Dropdown, setState];

Now we can use this just like we use hook, this is gonna give state and dropdown is gonna handle setting and unsetting and later we will use setState to programmatically update in addition to using dropdown

`use-dropdown-${label.replace("", "").toLowerCase()}`;

Above statement removes spaces

**How to handle Async code inside React**

API call for getting dropdown data from API

In SearchParam.js

Added useEffect

useEffect take place of several of lifecycle hooks like ComponentDidMount,ComponentDidUnMount

useEffect(()**=>**{

pet.breeds("dog").then(console.log,console.error)

})

useEffect is disconnected from when the render happending. What above function is doing is, this is schedule to render this function to run after rendering

An effect is run after every render (which happens after state changes.) (this is after the component is render it does API call) – So why this happens is you don’t want use to wait to see something( we need to show something)

To transaform list of objects to list of string – Map

***Problem :****Every time we type into the location useEffect requests a new set of breeds- Not corrected*

*We only request breeds when animal updates*

For above problem, rather than have useEffect after every single render, we say only run when these things change. For this declare things it depends on.

Here depends on animal

pet.breeds(animal).then(({ breeds }) **=>** {

**const** breedStrings = breeds.map(({ name }) **=>** name);

setBreeds(breedStrings);

}, console.error);

}, [animal]); --🡪 Here animal is added

[animal,setBreed,setBreeds])

-adding setBreeds- if any of these things change, re-render this effect

Summary about useEffect:

When location is changes-> it goes to useEffect and checks if animal,setBreed,setBreeds are changed since last useeffect was ran, If no-> Doesn’t schedule effect to run again.

Then select animal, then it sets (setBreeds([]);

setBreed("");)

and calls the API, get new breeds back and gonna re-render again

useEffect and useState are most common hooks used, then useRef

Try: in console: $r,

On submit:

**async** **function** requestPets() {

**const** { animals } = await pet.animal({

location,

breed,

type: animal

});

setPets(animals || []);

}

An async function is a function that is guaranteed to return a promise,

Await keyword tells wait here until this function completes then give back the data.

setPets(animals || []); - if animals don’t come backs sets empty array

**Reach Router**

**-Very accessibility focused. Like page change, it’ll focus on right page-(Focus maintaining is difficult)**

Reach router renders the thing that matches the most

Graphical user interface

Description automatically generated

For reach router- we can have multiple routers on page- like we can have a page and side nav that changes

Fun trick in Details.js

<pre>

<code>{JSON.stringify(props, null, 4)}</code>

</pre>

In page path you can see

View Props

{

"path": "/details/:id",

"id": "44131785",

"uri": "/details/44131785",

"location": {

"pathname": "/details/44131785",

"search": "",

"hash": "",

"href": "http://localhost:1234/details/44131785",

"origin": "http://localhost:1234",

"protocol": "http:",

"host": "localhost:1234",

"hostname": "localhost",

"port": "1234",

"state": null,

"key": "initial"

}

}

**Class Components**

**Can’t use hooks inside class components**

**Child receives props from parent and those props can’t be changed in child**

**This.state vs this.props**

**this.props**

* **Info from parent**

**In class we don’t have hooks so we don’t have anything to keep stateful things in class**

**So we use this.state**

**This.props is immutable**

**And this state is self contained within class. It’s mater of its own state.**

{

"presets": ["@babel/preset-react", "@babel/preset-env"],

"plugins": ["@babel/plugin-proposal-class-properties"]

}

**Above transforms code that works**

componentDidCatch

**static** getDerivedStateFromProps({media}){

}

**Above is special react method and must be static.**

**Get derived state from props, do some filtering and pass to component.**

**If you have derived state, can do this way**

handleIndexClick(event) {

this({

active: event.target.dataset.index

});

}

**Two things are wrong in above code:**

Dataset refers to data-index={index}(0,1,2,..)

Active-needs to be numbr, might be string(In react string is returned)- convert to int(+ before event converts to number)

Problem2:

This – window or undefined here- but need to be carousel

To make this correct

Method1:

**Graphical user interface

Description automatically generated**

**Method 2:**

**Turn to arrow function like below**

handleIndexClick = event **=>** {

this({

active: event.target.dataset.index

});

};

**Error Boundaries:**

Brian introduces error boundaries that allow for graceful capture of errors without an application blowing up. To implement the error boundaries, getDreivedStateFromError, and componentDidCatch React methods are introduced.

**Error Boundaries are one of two things Hooks cannot do.**

**Error Boundaries Graphical user interface

Description automatically generated**

**Above wouldn’t work , you can catch inside carousel but will not catch inside class itself**

export default **function** DetailsWithErrorBoundary(props) {

return (

<ErrorBoundary>

<Details {…props}/>

</ErrorBoundary>

);

}

**This will catch errors in details, how to get props to details??(** {…props}

Use spread operator when component doesn’t care about that props

componentDidUpdate is called when props or state change similar to useEffect

useEffect run when its dependencies update

getDerivedStateFromError

**gets called whenever there is error**

**ErrorB could be used to wrap a hook component**

**Context**

Context is like state, but instead of being confined to a component, it's global to your application. It's application-level state.

Context (mostly) replaces Redux. Well, typically. It fills the same need as Redux. I really can't see why you would need to use both. Use one or the other.

Imagine if we wanted to let the user choose a simple theme for the site. And we want to make that theme stick as the user navigates across different pages. This means the state has to live outside of the route where it's selected. We could use Redux for this, we could use React itself, or we're going to use context, to teach you what that looks like.

createContext is a function that returns an object with two React components in it: a Provide and Consumer

**A Provider is how you scope where a context goes. A context will only be available inside a provider**

**A Consumer is how you consume from above provider. A consumer accepts a function as a child and gives it the context which you can use. We won’t be using consumer directly a function called useContext will do that for us.**

**Easy to use with useContext hook**

**Context with Class**

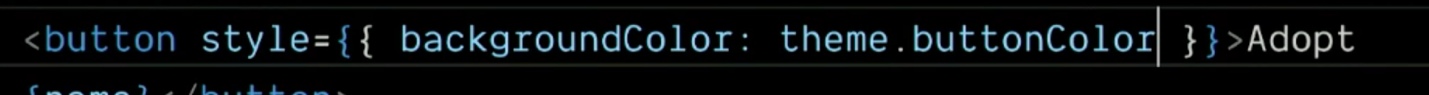
**Details is class**

**Ques: How could we pass multiple hooks to the context**

**At that point you wouldn’t want to use multiple hooks, give it an object like beloe**

**Graphical user interface, text

Description automatically generated**

****

**Hooks and context are totally different topics**

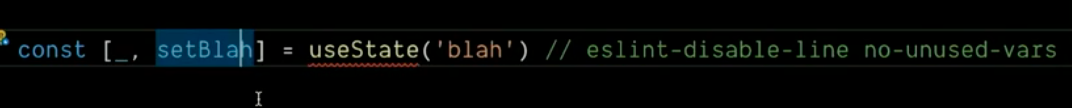
**We cannot use hooks inside class, that’s why we are using Consumer from ThemeContext**

<ThemeContext.Consumer>

**Error: if color of details page button doesn’t change;**

Brian demonstrates how utilizing a globally accessible variable within Context hooks allows a user to navigate away from the original page, and allow settings changed by the user to persist. The application is refactored to use Link tags instead of a tags to allow html history to navigate the pages. A question is asked about how to ignore a variable that's being returned from a hook.

Disable ESlint for a line

****

**Portals and Refs**

**Portal can be thought of as mount point for React app**

**const** Modal

**above code will mount a div inside of portal whenever Modal is rendered and unmount when modal in unrendered**

useEffect

**: We need to removed the div once modal is not being rendered- you can use a function inside of useEffect that cleans up**

We're also using a ref here via the hook useRef, Refs are like instant variables for function components. However on class you would use this.myVar to refer toa instance variable

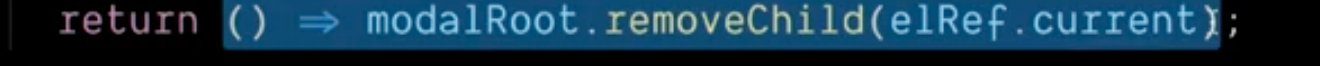
**Refs are containers of state that live outside a function’s closure state . That is anytime we refer elRef.current , its always refering the same element**

**Here elRef always points to same div**

**Graphical user interface, text, application

Description automatically generated**

[**https://btholt.github.io/complete-intro-to-react-v5/portals-and-refs**](https://btholt.github.io/complete-intro-to-react-v5/portals-and-refs)

**useEffect- if return a function , that’s a clean up function . Also runs below function when modal gets closed**

useEffect(() **=>** {

modalRoot.appendChild(elRef.current);

return () **=>** modalRoot.removeChild(elRef.current);

}, []);

**No dependencies and we wamt effect to run once**